

EXPRESS MAIL MAILING LABEL NUMBER EL521273096US

PATENT

10830.0084.NPUS00

APPLICATION FOR UNITED STATES LETTERS PATENT

for

**CLUSTER META FILE SYSTEM OF FILE SYSTEM CELLS
MANAGED BY RESPECTIVE DATA MOVERS OF
A NETWORK FILE SERVER**

by

Dinesh Venkatesh

Uday K. Gupta

BACKGROUND OF THE INVENTION

1. Field of the Invention.

The present invention relates generally to data storage, and specifically to file systems.

2. Description of Related Art.

Files are typically stored on disk in a hierarchical data structure called a file system. The file system includes a root directory and objects such as files, links, and subdirectories. The hierarchical arrangement was popularized in the UNIX operating system, and was also adopted in the Microsoft MS-DOS operating system for personal computers. The hierarchical arrangement survives today in various UNIX-based file systems, the Microsoft Windows operating system, and many other operating systems.

In a network environment, a network file server typically stores multiple file systems. A network client may access a specified file system using standard protocols such as the Network File System ("NFS") Protocol and the Common Internet File System ("CIFS") Protocol. NFS is described in Request for Comments ("RFC") 1094, Sun Microsystems, Inc., "NFS: Network File Systems Protocol Specification," March 1989, incorporated herein by reference. The CIFS protocol is described in Paul L. Leach and Dilip C. Narik, "A Common Internet File System," Microsoft Corporation, December 19, 1997, incorporated herein by reference.

A network storage system typically includes an array of disk drives and a storage controller for accessing the disk drives in response to client requests for file access. The storage controller is programmed with a number of software layers including a file system layer for mapping file names to logical storage locations, and a storage layer for

1 cache management and mapping of logical storage locations to physical storage locations
2 on disk. In a very high capacity system, the storage controller may have multiple
3 processing units, typically dedicated to file system layer functions or storage layer
4 functions. For example, the file system layer functions may be performed by multiple
5 commodity computers, and the storage layer functions may be performed by a cached
6 disk array, as described in Vahalia et al., U.S. Patent 5,893,140, issued April 6, 1999, and
7 entitled "File Server Having A File System Cache And Protocol For Truly Safe
8 Asynchronous Writes," incorporated herein by reference.

9 One problem with the use of multiple processing units for performing file system
10 layer functions is conflict between concurrent file access operations. For example, data
11 consistency problems may arise if concurrent access to a read/write file is permitted
12 through more than one processing unit. One solution to this problem, as described in the
13 Vahalia et al. U.S. Patent 5,893,140, is to store locking information in the data storage of
14 the network file server. Another solution to the problem, as described in Vahalia et al.
15 U.S. Patent 6,192,408 issued Feb. 20, 2001, incorporated herein by reference, is to assign
16 exclusive access rights for each read/write file system to a respective one of the
17 processing units. When a processing unit receives a client request for access to a
18 read/write file and finds that another processing unit has exclusive access rights to the file
19 system including the read/write file, it forwards the client request to the processing unit
20 having exclusive access rights to the file system, and the processing unit having exclusive
21 rights to the file system accesses the read/write file on behalf of the client. Still another
22 solution to the problem, as described in Xu et al. U.S. Patent 6,324,581 issued Nov. 27,
23 2001, is to assign exclusive metadata management rights for each read/write file system

1 to a respective one of the processing units. When a processing unit receives a client
2 request for access to a read/write file and finds that another processing unit has exclusive
3 metadata management rights to the file system including the read/write file, it forwards a
4 metadata request to the processing unit having exclusive metadata management rights to
5 the file system, and the processing unit having exclusive metadata management rights
6 responds by placing a lock on the read/write file and returning metadata of the file. The
7 processing unit that received the client request uses the metadata to formulate a data
8 access command for accessing the file data in the file system over a bypass data path that
9 bypasses the processing unit having exclusive metadata management rights to the file
10 system.

11 Although it is preferred to give a respective one of the processing units exclusive
12 rights to each read/write file system for preventing data consistency problems, there
13 could be a poor match between the number and size of the file systems stored in the
14 network file server and the number of processing units that may be assigned exclusive
15 access or management rights to the file systems. In other words, there is a need for an
16 efficient way of configuring exclusive rights of processing units in the network file server
17 to the file systems stored in the network file server.

19 SUMMARY OF THE INVENTION

20 In accordance with one aspect, the invention provides a method of accessing an
21 object in a meta file system stored in a network file server in a data network. The meta
22 file system includes a plurality of file system cells. The method includes a network client
23 sending a directory lookup request for the object to the network file server. The network

1 file server receives the directory lookup request, and in response, performs a directory
2 lookup for the object, and returns to the network client a file handle for the object. The
3 file handle includes an identifier of a file system cell including the object, and a pointer to
4 the object in the file system cell. The network client receives the file handle for the
5 object, and sends to the network file server a request for access to the object. The request
6 for access to the object includes the file handle for the object. The network file server
7 receives the request for access to the object, and in response, the network file server
8 extracts the file system cell identifier and the object pointer from the file handle included
9 in the request for access, and uses the file system cell identifier to find the file system cell
10 that includes the object, and uses the object pointer to find the object in the file system
11 cell.

12 In accordance with another aspect, the invention provides a method of accessing
13 an object in a meta file system stored in a network file server. The network file server
14 includes a cached disk array and a plurality of data mover computers for moving data
15 between the cached disk array and a data network. The meta file system includes a
16 plurality of file system cells. Each of the file system cells has a respective one of the data
17 mover computers assigned exclusive management of meta data of the file system cell.
18 The method includes storing a routing table in each of the plurality of data mover
19 computers. Each of the plurality of routing tables including an entry for each file system
20 cell. Each entry includes a respective file system cell identifier and associates the
21 respective file system cell identifier with a pointer to the respective one of the data mover
22 computers assigned exclusive management of metadata of the file system cell identified
23 by the respective file system cell identifier. In response to the network file server

1 receiving a request from the network for a file handle for a file, the network file server
2 produces a file handle for the file. The file handle contains a file identifier obtained from
3 the file system cell and a file system identifier for the file system cell containing the file.
4 At least one of the data mover computers receives a subsequent request from the network
5 for access to the file, and the request for access to the file includes the file handle. At
6 least one of the data movers responds to the request for access to the file by accessing the
7 routing table to obtain the pointer to the respective one of the data movers assigned
8 exclusive management of metadata of the file system cell containing the file in order to
9 obtain management of metadata of the file system cell containing the file in order to
10 access to the file.

11 In accordance with yet another aspect, the invention provides a network file
12 server including data storage for storing a meta file system. The meta file system
13 includes a plurality of file system cells. The network file server has at least one network
14 port coupled to the data storage for providing network clients with access to the meta file
15 system in the data storage. The network file server is programmed for receiving, from a
16 network client, a directory lookup request for an object in the meta file system, and in
17 response, performing a directory lookup for the object, and returning to the network
18 client a file handle for the object, the file handle including an identifier of a file system
19 cell including the object, and a pointer to the object in the file system cell. The network
20 file server is also programmed for receiving, from the network client, a request for access
21 to the object, the request for access to the object including the file handle for the object,
22 and in response to receipt of the request for access to the object, extracting the file system
23 cell identifier and the object pointer from the file handle included in the request for

1 access, using the file system cell identifier to find the file system cell that includes the
2 object, and using the object pointer to find the object in the file system cell.

3 In accordance with yet still another aspect, the invention provides a network file
4 server including a cached disk array and a plurality of data mover computers for moving
5 data between a data network and a meta file system stored in the cached disk array. The
6 meta file system includes a plurality of file system cells. Each of the file system cells has
7 a respective one of the data mover computers assigned exclusive management of
8 metadata of the file system cell. The network file server is programmed for storing a
9 routing table in each of the data mover computers. The routing table in each of the data
10 mover computers includes an entry for each file system cell, each entry including a
11 respective file system cell identifier and associating the respective file system cell
12 identifier with a pointer to the respective one of the data mover computers assigned
13 exclusive management of metadata of the file system cell identified by said respective
14 file system cell identifier. The network file server is also programmed for responding to
15 receipt of a request from the network for a file handle for a file by producing a file handle
16 for the file, the file handle containing a file identifier obtained from the file system cell
17 and a file system identifier for the file system cell containing the file. At least one of the
18 data mover computers is programmed for receiving a subsequent request from the
19 network for access to the file, the request for access to the file including the file handle,
20 and for responding to the request for access to the file by accessing the routing table in
21 the at least one of the data mover computers to obtain the pointer to the respective one of
22 the data movers assigned exclusive management of metadata of the file system cell
23 containing the file in order to obtain management of metadata of the file system cell

1 containing the file in order to access to the file.

3 BRIEF DESCRIPTION OF THE DRAWINGS

4 Other objects and advantages of the invention will become apparent upon reading
5 the detailed description with reference to the drawings, in which:

6 FIG. 1 is a block diagram of a file system as seen by a user or application
7 program;

8 FIG. 2 is a block diagram of a meta file system for implementing the user-visible
9 file system of FIG. 1;

10 FIG. 3 is a flow chart of a procedure for creating a new meta file system from an
11 original file system or an original meta file system;

12 FIG. 4 is a flow chart of a meta file system manager routine for accessing a root
13 directory of a file system cell in the meta file system of FIG. 2;

14 FIG. 5 is a flow chart of a meta file system manager routine for accessing a
15 directory in a file system cell having external links;

16 FIG. 6 is a flow chart of a meta file system manager routine for crash recovery of
17 a meta file system;

18 FIG. 7 is a flow chart of a meta file system manager routine for crash recovery of
19 a specified file system;

20 FIG. 8 is a block diagram of a data processing system including a network file
21 server having a plurality of data mover computers, each of which manages a respective
22 file system cell of the meta file system of FIG. 2;

1 FIG. 9 is a partial view of the data network of FIG. 8, showing in detail one of the
2 data mover computers in the network file server of FIG. 8;

3 FIG. 10 is a block diagram showing the management of metadata for a file in a
4 data mover that owns the file and a data mover that is secondary with respect to the file;

5 FIG. 11 is a flow chart showing how the data mover computers in the data
6 processing system of FIG. 8 manage their respective cells in the meta file system of FIG.
7 2;

8 FIG. 12 is a flow chart showing a network protocol for file access in accordance
9 with one aspect of the present invention;

10 FIG. 13 is a flow chart showing how information in a file handle is used to access
11 an entry in a routing table for routing a file access request in the network file server of
12 FIG. 8;

13 FIG. 14 is a flow chart showing how the procedure of FIG. 13 can be used to
14 resolve hidden references to file system cells during processing of a file access request;
15 and

16 FIG. 15 shows an alternative arrangement for the routing table, in which the
17 routing table includes pointers to file system cell objects.

18 While the invention is susceptible to various modifications and alternative forms,
19 a specific embodiment thereof has been shown by way of example in the drawings and
20 will be described in detail. It should be understood, however, that it is not intended to
21 limit the form of the invention to the particular form shown, but on the contrary, the
22 intention is to cover all modifications, equivalents, and alternatives falling within the
23 scope of the invention as defined by the appended claims.

DESCRIPTION OF ILLUSTRATIVE EMBODIMENTS

With reference to FIG. 1, there is shown a block diagram of a file system 21 as seen by a user or application program. The file system is a hierarchy of data objects such as directories, files, and links. The data objects reside in logical blocks of data storage. Typically a contiguous range of logical block numbers of data storage is allocated to store the file system objects and additional objects that might be added to the file system.

Typically a user or application program accesses an object in the file system by providing a path name for the object. Given the path name, an operating system program known as a file system manager searches the file system hierarchy for the specified object. The file system manager accesses the file system from an entry point 22, which is the beginning of a root directory 23 named "A:".

The root directory has a list of directory entries, each of which points to a file system object. Each entry, for example, includes a name of the object pointed to by the entry, and an object type attribute indicating whether the object is a subdirectory, a file, or a link. For example, the root directory 23 lists a subdirectory 24 named "E", a subdirectory 25 named "F", a subdirectory 26 named "G", a subdirectory 27 named "H", a file 28 named "I", and a file 29 named "J". Each subdirectory may also list subdirectories, files, and links. For example, the subdirectory 24 named "E" lists a subdirectory 29 named "K", and the subdirectory named "K" lists a subdirectory 30 named "L". The subdirectory 25 named "F" lists a file 31 named "N". The subdirectory 26 named "G" lists a link 32 named "N". The link 32 includes a pointer to the file 31 named "N".

1 than 100 gigabytes. Because file system size limits have increased to at least 800
2 gigabytes, and users frequently add rather than delete new data and new files, users and
3 system administrators may not be prepared for the delay that should be expected for
4 recovery from a system crash.

5 The present invention solves the problem of large file systems by recognizing that
6 it is possible for users, and application programs such as backup programs, to have a
7 view of the file system that is different from the file system organization seen by the
8 operating system and crash recovery programs. The user and application programs can
9 see one very large file system, and the operating system and crash recovery programs can
10 see a collection of smaller file systems that include all of the objects in the one very large
11 file sytem. Because file systems traditionally have been managed at the operating system
12 level, the smaller file systems will be considered as real file systems, and the collection of
13 file systems seen by the user as one file system will be referred to as a "meta file system."
14 The file systems in the meta file system will be referred to as "file system cells." The
15 present invention also provides a meta file system manager in the operating system for
16 permitting users and application programs to access the meta file system as if the meta
17 file system were one large conventional file system.

18 The file system of FIG. 1 could be broken down into file system cells in various
19 ways. Specifically, each subdirectory in the file system 21 is a directory of a file
20 subsystem in the file system 21. As shown in FIG. 1, the file system 21 includes a file
21 subsystem 35 named "E", a file subsystem 36 named "K", a file subsystem 37 named
22 "L", a file subsystem 38 named "F", a file subsystem 39 named "G", and a file subsystem
23 40 named "H". Each of these file subsystems could become a file system cell in a meta

1 file system having the same appearance to a user or application program as the file
2 system 21 shown in FIG. 1.

3 With reference to FIG. 2, there is shown a specific example of a meta file system
4 having the same appearance to the user or application program as the file system of FIG.
5 1. As shown in FIG. 2, the file system cells of the meta file system consist of a file
6 system 41 named "A:", a file system 42 named "B:", a file system 43 named "C:", and a
7 file system 44 named "D:". The root directory 45 named "A:" in FIG. 2 is similar to the
8 root directory 23 named "A:" in FIG. 1 to the extent that it has entries including object
9 names E, F, G, H, I, and J. In the directory 45 named "A:" in FIG. 2, however, the
10 objects named "E" and "F" have become external links to the root directories 47 and 54
11 of the file systems 42 and 43 named "B:" and "C:", respectively. For example, each of
12 the entries in the root directory 45 named "A:" has an attribute flag (shown as an asterisk)
13 indicating to the meta file system manager whether or not the entry is for an external link.
14 The root directory entry containing the name "E" points to an external link 46 to the file
15 system cell 42 named "B:", and the root directory entry containing the name "F" points to
16 an external link 53 to the file system cell 43 named "C:". The external links 46 and 53,
17 for example, are files that are hidden from the view of the user or application program, so
18 that they do not appear to the user or application program be files in the root directory 45,
19 and instead the items named "E" and "F" in the root directory 45 appear to the user or
20 application program to be the directory 47 named "B:" and the directory 54 named "C:",
21 respectively. Moreover, to the user or application program, the directory 47 named "B:"
22 and the directory 54 named "C:" appear to be subdirectories, and not root directories.
23 Such external links, which are hidden from the view of the user or application program

1 and which link the file system cells to form the meta file system hierarchy, will be
2 referred to as "direct external links."

3 The subdirectory 57 named "G" in the meta file system of FIG. 2 is similar to the
4 subdirectory 26 named "G" in the file system of FIG. 1 except that the entry named "M"
5 in the subdirectory 57 named "G" in the meta file system has an attribute flag (shown as
6 an asterisk in front of the M) set to indicate that the entry named "M" is for an external
7 link. This subdirectory entry named "M" points to a link 32 to the file 56 named "N",
8 which appears in the file system cell 43 named "C:". In this case, the link 32 is not
9 hidden from the user or application. Such a user-visible external link will be referred to
10 as an "indirect external link."

11 The file cells themselves may have a hierarchical arrangement. For example, the
12 file system cell 42 named "B:" has a subdirectory 49 named "K" including the entry
13 containing the name "L" pointing to an external link 50 to the root directory 51 of the file
14 system cell 44 named "D:". Each of the file system cells in the meta file system cell
15 hierarchy includes respective file system meta data 58, 48, 52, 55.

16 The original file system 21 named "A:" in FIG. 1 has become a file system cell 41
17 named "A:" in the meta file system of FIG. 2. It is desired that the file system cell "A:"
18 and the other file system cells comply with the conventional format of a file system. In
19 this case, the links in an original file system must be converted to external links that do
20 not cause errors when interpreted by a conventional file system manager. In a UNIX-
21 based file system, for example, a link in an original file system includes a pointer to an
22 inode of the target object. If the target object has been moved to another file system cell,
23 then this pointer will be invalid. Nor can the pointer simply be changed to point to a new

inode for the target, because the pointer values are inode numbers that are local to each file system. Therefore, it may be desirable to put any link in the original file system into a form that is recognized by the conventional file system manager as an empty target, so that the conventional file system manager will not attempt to access a target for the link. Nevertheless, an external link needs to contain something that can be interpreted by the meta file system manager as a pointer to the new location of the target in the new file system cell. Preferably this pointer is not in the form of a logical block number or inode number in the new file system cell, because then the logical block number or inode number of the target may change if and when the new file system cell is reorganized. The new file system may be reorganized, for example, by converting a file subsystem containing the target object into a new file system cell.

One solution to the problem of links in the user-visible file system is to use the user-visible pathname of the target object as the target object specification in any external link converted from an internal link appearing in the user-visible file system. For example, the link 32 to the file named "N" in FIG. 2 would include the user-visible pathname "A:\F\N." Therefore, the target object specification need not be changed in response to any reorganization of the meta file system that does not affect the user-visible file system. Nevertheless, the use of the user-visible pathname for the target object specification will introduce some search delay for finding the target, and therefore the liberal use of links in the user-visible path specification should be discouraged.

In contrast, the direct external links (such as links 46 and 53 in FIG. 2) that are created during meta file system creation or reorganization may contain a target specification that is simply the name of the new file system cell (such as "B:" or "C:" for

1 the respective external links 46 and 53). Although these file system cell names are
2 invisible to the user or application, they define the meta file system hierarchy and
3 therefore should be maintained by appropriate modification if and when they would be
4 affected by any file system reorganization.

5 Due to the use of external links in the file system cells of the meta file system, it
6 is desirable to include a meta file system flag (such as the flag 60 in the file system
7 metadata 58) in the file system metadata of any file system cell that includes external
8 links. In the meta file system of FIG. 2, for example, the meta file system flag 60 is set, a
9 similar flag (not shown) in the file system metadata 48 is set, a similar flag (not shown) in
10 the file system metadata 52 is not set, and a similar flag (not shown) in the file system
11 metadata 55 is not set. When set, such a meta file system flag would indicate that access
12 of the file system cell with a conventional file system may result in unexpected behavior
13 or a report of an empty link. For example, an attempted execution or access of the link
14 file could result in an immediate return or a user message "target not found." If the user
15 or application would desire to access the target objects of the external links, then the file
16 system cell should be accessed with a meta file system manager, as further described
17 below with reference to FIGS. 4 to 7.

18 With reference to FIG. 3, there is shown a flow chart of a procedure for creating a
19 new meta file system from an original file system or from an original meta file system.
20 In a first step 61, a subdirectory is selected or created in the original file system or in a
21 file system cell of the original meta file system. Next, in step 62, the file subsystem of
22 the selected subdirectory is converted into a new file system cell. The subdirectory
23 becomes the root directory of the new file system cell. In step 63, an external link is

1 created between the subdirectory's parent directory and the root directory of the new file
2 system cell. This external link is placed at the same level in the hierarchy of the meta file
3 system as subdirectory's parent directory. Finally, in step 64, the new meta file system is
4 searched for any internal links between the new file system cell and the other file system
5 cell or cells in the new meta file system. If any such internal link is found, it is converted
6 to an external link.

7 With reference to FIG. 4, there is shown a flow chart of a routine in a meta file
8 system manager for accessing the root directory of a file system cell. In a first step 71,
9 the meta file system manager inspects the file system meta data to determine whether the
10 meta file system flag is set. If the meta file system flag is not set, then execution
11 continues to step 72 to use conventional file system manager routines to access the file
12 system. Otherwise, if the meta file system flag is set, then execution branches from step
13 71 to step 73. In step 73, the meta file system manager uses some special routines to
14 recognize and interpret external links to objects in other file system cells in the meta file
15 system. Otherwise, the meta file system manager uses conventional file system manager
16 routines to access the file system.

17 With reference to FIG. 5, there is shown a flow chart of a meta file system
18 manager routine for accessing a directory in a file system cell having external links. In a
19 first step 81, the meta file system manager obtains the next entry in the directory. If an
20 entry is not found, as tested in step 82, execution returns. Otherwise, execution continues
21 from step 82 to step 83. In step 83, the meta file system manager checks an attribute for
22 the directory entry to determine whether the directory entry points to an external link. If
23 not, then execution branches from step 83 to step 84 to use conventional file access

1 routines of a conventional file system manager to interpret and access the object. After
2 step 84, execution loops back to step 81.

3 If in step 83 the meta file system manager determines that the directory entry is
4 for an external link, execution continues from step 83 to step 85. In step 85, the meta file
5 system manager sets the kernel context to the file system at the head of the path
6 specification for the link target and continues interpretation of the path specification until
7 the link target is found in the meta file system. The user file system context, however, is
8 not changed. For a link in the user-visible file system, the path specification is the user-
9 visible path specification. Otherwise, for a direct link that does not appear in the user-
10 visible file system, the path specification is simply the root directory name for another
11 file system cell.

12 After step 85, execution continues to step 86. In step 86, if a target is not found,
13 then an error message is returned. Otherwise, execution loops back from step 86 to step
14 83, to test whether the target is itself an external link. Execution then continues until all
15 of the entries in the directory have been accessed, or until step 84 terminates directory
16 access before all of the entries in the directory have been accessed.

17 With reference to FIG. 6, there is shown a flow chart of a meta file system
18 manager routine for crash recovery of a meta file system. In step 91, the meta file system
19 performs a repair of each file system cell in the meta file system. For example, the meta
20 file system manager scans a list of the file system cells in the meta file system. For each
21 file system cell, the meta file system manager calls a repair subroutine as shown in FIG.
22 7.

1 In the repair subroutine of FIG. 7, the meta file system manager inspects the file
2 system metadata for the file system cell to be repaired. If the consistency flag is set, as
3 tested in step 101, then the subroutine is finished, and execution returns. Otherwise,
4 execution continues to step 102 to scan the file system cell for inconsistency with a log of
5 updates, and to repair any inconsistency that is found. Then execution returns.

6 As described above, one advantage of a meta file system is to reduce the time for
7 recovery of a large user-visible file system. Another advantage of the meta file system is
8 to permit multi-processor management of a file system by concurrently performing file
9 system access operations in multiple file system cells of the meta file system.

10 With reference to FIG. 8, there is shown a block diagram of a data processing
11 system including a network file server 110 having a plurality of data mover computers
12 115, 116, 117, each of which manages a respective file system cell of the meta file
13 system of FIG. 2. The network file server 110 is connected by a data network 111 to a
14 number of clients including work stations 112, 113. The data network 111 may include
15 any one or more network connection technologies, such as Ethernet, and communication
16 protocols, such as TCP/IP or UDP. The work stations 112, 113, for example, are
17 personal computers.

18 The preferred construction and operation of the network file server 110 is further
19 described in Vahalia et al., U.S. Patent 5,893,140 issued April 6, 1999, incorporated
20 herein by reference. The network file server 110 includes a cached disk array 114. Such
21 a network file server 110 is manufactured and sold by EMC Corporation, 35 Parkwood
22 Dr., Hopkinton, MA 01748.

1 The network file server 110 is managed as a dedicated network appliance,
2 integrated with popular network operating systems in a way, which, other than its
3 superior performance, is transparent to the end user. The clustering of the data movers
4 115, 116, 117 as a front end to the cached disk array 114 provides parallelism and
5 scalability. Each of the data movers 115, 116, 117 is a high-end commodity computer,
6 providing the highest performance appropriate for a data mover at the lowest cost. The
7 data movers may communicate with each other over a dedicated dual-redundant Ethernet
8 connection 118. The data mover computers 115, 116, and 117 may communicate with
9 the other network devices using standard file access protocols such as the Network File
10 System (NFS) or the Common Internet File System (CIFS) protocols, but the data mover
11 computers do not necessarily employ standard operating systems. For example, the
12 network file server 110 is programmed with a Unix-based file system that has been
13 adapted for rapid file access and streaming of data between the cached disk array 114 and
14 the data network 111 by any one of the data mover computers 115, 116, 117.

15 The meta file system can be organized so that the file system cell named "A:"
16 does not contain any user or application files. In this case, the file system cell named
17 "A:" is essentially a repository for metadata of the user-visible file system, and
18 information about the configuration of the meta file system. For example, the root
19 directory of the file system cell named "A:" shows how objects in the user-visible root
20 directory are apportioned among other file system cells containing user-visible files. In
21 the example of FIG. 8, the file system cell "A:" is located in data storage 119 in the
22 cached disk array 114, and each of the data movers 115, 116, 117 has shared read-write
23 access to the file system cell "A:" in the data storage 119. Therefore, each of the data

1 movers has non-blocking concurrent read-write access to the information needed for
2 directing a file system access received by any of the data movers to the file system cells
3 (B:, C:, and D:) containing user or application files. Moreover, as shown in FIG. 8, the
4 other file system cells (B:, C:, and D:) are also located in respective data storage 120,
5 121, 122 in the cached disk array 114. Each of the data movers 115, 116, 117 has read-
6 write access to each of the file system cells. However, the data mover 115 exclusively
7 manages access to the file system cell "A:" and the file system "B:" (i.e., it is the "A:
8 owner" and the "B: owner"), the data mover 116 exclusively manages access to the file
9 system cell "C:" (i.e., it is the "C: owner"), and the data mover 117 exclusively manages
10 access to the file system cell "D:" (i.e., it is the "D: owner"). The initial assignment of file
11 system ownership is done at a control station 123 during configuration time.

12 As shown in FIG. 8, the network file server 110 includes offline storage 130
13 coupled to the cached disk array. The offline storage 130 may include tape or optical
14 disk storing back-up or offline versions of certain files in the file systems A:, B:, C:, or
15 D:. If data of a particular file presently resides in disk storage of the cached disk array
16 114, then the file is said to be "online." If data of a particular file does not presently
17 reside on disk storage in the cached disk array 114 but the data resides in the offline
18 storage 130, then the file is said to be "offline."

19 Referring to FIG. 9, there is shown a block diagram of software structure that is
20 replicated in each data mover 115. The software structure includes modules 141 and 142
21 for the Network File System (NFS) file access protocol (FAP) and the Common Internet
22 File System (CIFS) file access protocol (FAP), a Virtual File System (VFS) 143, and a
23 Unix-based file system (UxFS) 144. The Virtual File System (VFS), which is an

1 industry-standard back-end file system switch, interfaces with the UxFS physical file
2 system 144. VFS translates NFS Common File System requests. (The NFS Common
3 File System requests in themselves are translations of NFS requests to the intended
4 physical file storage devices.) UxFS accesses a file system buffer cache 145 during data
5 transfers between the data network 111 and UxFS files 149 in the cached disk array 110.
6 UxFS also writes updates to a log file 150 just before committing the updates to the UxFS
7 files 149, for example, upon closing a file after a write operation.

8 As shown in FIG. 9, a meta file system manager 146 is integrated with the VFS
9 module 143. For example, the VFS module 143, including the meta file system manager
10 146, is read from a machine readable program storage device such as a floppy disk 147,
11 and loaded into local disk data storage 151 of the data mover 115. As introduced above
12 with reference to FIG. 4, the meta file system manager 146 is invoked by file system
13 access calls originally intended for the file system manager access routines in the UxFS
14 144.

15 The meta file system manager 146 includes a routine for accessing the root
16 directory of a file system cell. If the meta file system flag for the file system cell is not
17 set, conventional file manager routines in the file system manager are called to access the
18 file system. If the meta file system flag is set, the meta file system manager calls meta
19 file system routines to recognize and interpret external links to objects in other file
20 system cells in the meta file system.

21 The meta file system manager 146 also includes a routine for accessing a
22 directory in a file system cell having external links. The routine checks an attribute
23 associated with each directory entry to determine whether the directory entry points to an

1 external link. If so, the routine sets the kernel context to the file system at the head of the
2 path specification for the link target and continues interpretation of the path specification
3 until the link target is found in the meta file system.

4 In operation, a client makes a request for access to a file stored in a cached disk
5 array 110 over the data network 111 using a standard protocol such as the NFS protocol
6 or the CIFS protocol. When using the NFS protocol, for example, a client first issues an
7 NFS lookup request including a path to a file and the filename for the file to be accessed.
8 The lookup request returns a file handle for a file entry corresponding to the file. The
9 client uses the file handle in subsequent requests for access to the file. These subsequent
10 requests are interpreted by the NFS routines 141 and forwarded to the meta file system
11 manager 146.

12 The CIFS protocol is based on the connection-oriented Transmission Control
13 Protocol (TCP). A file system can be shared amongst CIFS clients by forwarding
14 requests between a group of CIFS servers. The group of CIFS servers appears to the
15 CIFS clients as a single file server and provides enhanced data availability, reliability and
16 storage capacity. When using the CIFS protocol, for example, the client sends a series of
17 commands to the network file server, including a NEGOTIATE command to initiate
18 communication with the network file server, a SESSION_SETUP command for
19 verification of the client's name and credentials, a TREE_CONNECT command for
20 access to specified file system, and then a number of file access commands such as
21 OPEN, READ, and CLOSE, followed by a TREE_DISCONNECT command. The CIFS
22 server routines 142 receive the CIFS request from the client and forward the request
23 through the Virtual File System (VFS) to the meta file system manager 146. For

example, in the TREE_CONNECT command, the client transmits the name of the file system that the client wants to access, and the server returns a tree id (Tid) that the client uses in subsequent commands to refer to the file system. In the OPEN command, the client transmits the name of the file, relative to the Tid, that the client wants to open. The network file server returns a file handle, referred to as the file id (Fid). In the READ command, the client supplies the Tid, Fid, file offset, and number of bytes to read.

In order to process a file access request, the meta file system manager 146 may need data or metadata of a file system object. In such a case, the meta file system manager 146 accesses a file system routing table 148 in order to determine the data mover that owns the file system object. If the data mover 115 owns the object, then the object is local, and the meta file system manager 146 accesses data or metadata of the file system object through UxFS 144. If the data mover 115 does not own the file system object, then the object is said to be "remote" and the data mover 115 is said to be "secondary" with respect to the object. When the file system object is remote, the meta file system manager 146 accesses data or metadata of the file system object through a Multiplex File System (MPFS) 152.

FIG. 10 shows how the MPFS 152 of the data mover 115 accesses data or metadata of a file in the file system "C:" owned by the data mover 116. In this case the file's metadata 153 is cached inside UxFS 159 in the data mover 116, and the metadata log is updated only by a synchronous write, so that the metadata 158 is only known to the owner, data mover 116. Therefore, the metadata 158 is sent to the secondary data movers (such as the data mover 115) if they also want to operate on the file. A version number 154 associated with the metadata of each file is used to guarantee that every data mover

1 always uses the most up-to-date version of the metadata to access the file. Metadata is
2 also cached on secondary data movers to improve performance. This metadata 155 is
3 cached inside MPFS 152 in the secondary data mover 115. This metadata 155 also has
4 an associated version number 156. Every time the metadata is changed on a data mover,
5 the version number associated with that metadata on that data mover is increased by one.
6 During a commit or close operation, new metadata is written back from the owner's
7 metadata cache 153 to metadata storage 158 of the file system 121 in the data storage
8 device (such as the cached disk array 114). To avoid a data security problem, the
9 metadata 158 in the file system 121 is always written back to data storage after the
10 corresponding data 157 has been updated.

11 To access the file in the file system "C:", the secondary data mover 115 sends a
12 file lock request, including its metadata version number 156 for the metadata of the file,
13 to the data mover owner 116 of the file. The data mover replies with a lock grant or lock
14 denied message and also returns new metadata if the metadata version number 156 from
15 the secondary data mover 115 is different from its metadata version number. When the
16 file lock has been granted and the secondary data mover has a current version of the file's
17 metadata, the secondary data mover 115 may directly access the file data 157 in the
18 storage 121 of the cached disk array, over a data path that bypasses the data mover owner
19 116 of the file. When the secondary data mover 115 has finished accessing the file, it
20 sends a confirmation of a release of the file lock, including any change to the file
21 metadata that may have occurred due to the file access. Further details regarding the
22 operation of the MPFS 152 are found in the above-cited Vahalia et al. U.S. Patent
23 6,324,581.

1 With reference to FIG. 11, there is shown a flow chart of a procedure by which
2 the data mover computers in the data processing system of FIG. 8 manage their
3 respective file system cells in the meta file system of FIG. 2. In a first step 161, the meta
4 file system is configured so that each data mover owns a set of file system cells in the
5 meta file system. Then in step 162, each data mover performs locking, access, logging of
6 updates, and commitment of updates upon the file system cells that it owns. If a data
7 mover receives from a network client a request for access to a meta file system object in a
8 file system cell that the data mover does not own, then the data mover forwards the
9 request to the data mover that owns the file subsystem cell, and also returns any reply to
10 the network user having originated the request.

11 In the preferred implementation (as described above with reference to FIG. 11), a
12 data mover that does not own a file system cell to be accessed sends a lock request to the
13 data mover that owns the file system cell to be accessed, and the data mover that owns
14 the file system cell to be accessed returns a grant of a lock on an object in the file system
15 cell to be accessed. The data mover that does not own the file system cell to be accessed
16 can then access the object directly in the cached disk array for satisfying the client's
17 request, without passing object data through the data mover that owns the file system cell
18 to be accessed. In this alternative arrangement, however, it is still preferred for the data
19 mover that owns the file system cell to maintain and update the meta data for the objects
20 in the file system cell that it owns. Once the access for the client is completed, the data
21 mover not owning the file system cell sends a notification of a release of the lock to the
22 data mover that owns the file system that is accesses.

1 In step 163, each data mover monitors loading upon the file system cells that it
2 owns, and upon reaching a high loading level, the data mover polls other data movers for
3 their loading levels. If a data mover having a sufficiently lower loading level is found,
4 then load balancing is performed by shifting file subsystems or file system cells from the
5 more heavily loaded data mover to the less heavily loaded data mover. A shifted file
6 subsystem becomes a new file system cell in the meta file system. A shift of a file
7 system cell is a change in data mover ownership of the file system cell. In either case, the
8 shift is performed by adding or changing linkages in the meta file system hierarchy, and
9 is essentially a change in pointers to file data. The file data itself need not be moved in
10 the storage of the cached disk array.

11 In step 164, if a data mover crash occurs, then once the data mover having
12 crashed is re-booted, it performs crash recovery upon the file system cells that it owns. In
13 step 165, if a system crash occurs, then each data mover in the network file server
14 performs crash recovery upon the file system cells that it owns. This completes the flow
15 chart in FIG. 11 of the procedure for maintenance of the meta file system in the network
16 file server in FIG. 8.

17 FIG. 12 shows a flow chart of a network protocol for file access in accordance
18 with an aspect of the present invention. In a first step 166, a network client sends to the
19 network file server a directory lookup request to find a specified object in a meta file
20 system.

21 In step 167, the network file server receives the directory lookup request, and in
22 response performs a directory lookup, finds the specified object in a file system cell in the
23 meta file system, and returns a file handle for the object. The file handle includes an

1 identifier of the file system cell including the object, and a pointer to the object in the file
2 system cell.

3 In step 168, the network client receives the file handle from the network file
4 server, and uses it in a request to the network file server for access to the object.

5 In step 169, the network file server receives the file handle, and extracts the file
6 system cell identifier and the object pointer from the file handle. The network file server
7 uses the file system cell identifier to find the file system cell that includes the object, and
8 uses the object pointer to find the object in the file system cell.

9 FIG. 13 shows how information in a file handle 171 is used to access an entry in
10 the file system routing table (148 in FIG. 9). In this example, the meta file system
11 manager (146 in FIG. 9) responds to a directory lookup request in such a way that the file
12 handle 171 returned to the client in response to a directory lookup request specifying a
13 path name and a file system object name includes a file system ID 172 of the file system
14 cell including the file system object in addition to a pointer 174 to the object in the file
15 system cell. In particular, in response to a network client's directory lookup request
16 specifying a meta file system path name and a name of an object in the meta file system,
17 the meta file system manager (146 in FIG. 9) translates the directory lookup request to a
18 file system cell identifier, and a UxFS directory lookup request including the name of a
19 path in the file system cell, and the name of the object in the file system cell. UxFS
20 responds to the directory lookup request by returning UxFS information about the object,
21 including a pointer to the object in the file system cell. The meta file system manager
22 146 receives the UxFS information 173, and creates the file handle 171 by appending, to
23 the UxFS information 173, a file system ID 172 of the file system cell, and also a file

1 type 175 including an indication of whether the object is online or offline. The file type
2 175 may include additional descriptive information about the file, such as the file format
3 (e.g., text, graphics or video) or identification of a particular application program
4 associated with the file. The meta file system manager 146 returns this file handle 171 to
5 the network client.

6 In a specific implementation, for example, the UxFS information returned to the
7 meta file system manager is a FILE structure including a pointer to a buffer for
8 temporarily storing contents of the file, a count of the number of characters stored in the
9 buffer, a pointer to the next character position in the buffer, flags for storing the access
10 mode (read and write permissions for owner, group and others) and a file descriptor
11 including a file identifier and a generation count used to validate the file identifier. The
12 meta file system file handle 171 is 128 bits with a 32-bit file type, a 32-bit file system
13 identifier of the file system cell, a 32-bit UxFS file identifier pointing to the object in the
14 file system cell, and a 32-bit UxFS generation count. The maximum number of bits in
15 the file handle 171 is dependent on the maximum number of bits that the network
16 protocol allows for the file handle.

17 In an embodiment using the NFS protocol, the meta file system file handle is
18 limited by the 32 or 64 bytes allowed by the NFS protocol. Thus, the file handle can be
19 expanded in other embodiments from the sixteen bytes in this specific implementation to
20 include more UxFS information or increase the maximum possible number of file system
21 cells in the meta file system. However, the present invention is not limited to use of the
22 NFS protocol, and other protocols, such as the CIFS protocol, could be used for sending a
23 file handle between the network file server and a network client.

1 From FIG. 13, it is seen that by including the file ID 172 of the file system cell in
2 the file handle 172 sent to and received from the network client, it is very easy for the file
3 system manager to access a file system routing table to direct a file system access request
4 from a client to the data mover owner of the object to be accessed. Because the file
5 system routing table 148 need only include one entry for each file system cell in the meta
6 file system, it is relatively compact and is easily replicated in each of the data movers.

7 Although the flow chart in FIG. 13 has been described with reference to accessing
8 an object in response to a client providing a file handle including a file system ID of a file
9 system cell including the object, the procedure of FIG. 13 is also applicable to a directory
10 lookup request in which the file handle is for the directory but the information sought in
11 the directory is found in another file system cell that may be referenced in the directory.
12 The management of this situation is shown in FIG. 14.

13 In step 191 of FIG. 14, execution branches to step 192 if access to the object in
14 step 180 or step 182 encounters a reference to another file system cell. For example, a
15 directory lookup may encounter a hidden link specifying another file system cell. In this
16 case, in step 192, steps 178, 179, and 180 or 181 are repeated to extend the directory
17 lookup into the file system cell specified by the hidden link. When there is no further
18 references to additional file system cells, execution continues from step 191 to 193. In
19 step 192, if the object sought has been found, execution returns normally. Otherwise, if
20 the object sought cannot be found, then execution returns with an error code indicating
21 that the object has not been found.

22 FIG. 15 shows an alternative construction for a file system routing table 201. In
23 this example, the routing table associates each file system cell ID with a respective

1 pointer to a file system cell object 202, 203, 204, etc. Each file system cell object, such
2 as the object 202, includes a local/remote flag 205 indicating whether or not the local data
3 mover owns the file system cell, a data mover owner IP address pointing to the data
4 mover that owns the file system cell, and an identifier 207 of a communication protocol
5 for communicating with the data mover that owns the file system cell.

6 In view of the above, file system cells are linked together to form a meta file
7 system that appears to a user or application program to be a single file system. Each file
8 system cell may have a conventional file system format, and can be indistinguishable
9 from a conventional file system except for information, such as directory entry attributes,
10 indicating one or more external links to other file system cells. These external links may
11 include direct links that are hidden from the user or application program and define a
12 hierarchy of the meta file system cells, and indirect links that appear in the user-visible
13 file system. Preferably the indirect links include a target specification that is the name of
14 a path in the user-visible file system so that the target does not change during any re-
15 organization of the meta file system. The meta file system substantially reduces crash
16 recovery time because each file system cell functions as a consistency unit that can be
17 repaired only if needed. The meta file system also permits the file system cells to be
18 accessed concurrently by multiple processors in a file server. Each processor can be
19 assigned exclusive management of a set of file system cells. As objects such as files are
20 added to the file system cells, each processor can monitor loading upon its file system
21 cells, and respond to a high loading condition by polling the other processors to find a
22 less heavily loaded processor. If a less heavily loaded processor is found, the meta file
23 system can be reorganized to perform load balancing by shifting the load from the more

1 heavily loaded processor to the less heavily loaded processor. In order to determine
2 easily which processor is assigned to manage access to a file, the file handle used in a
3 network file access protocol includes an identifier of the file system cell and a pointer to
4 the file in the file system cell. A routing table in each processor associates each file
5 system cell identifier with a pointer to the processor assigned to access the file.